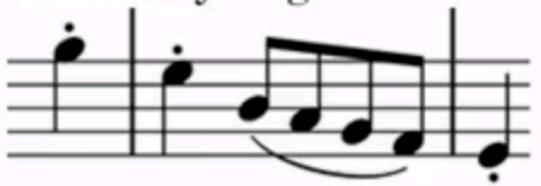
RECURRENT NEURAL NETWORK

NEURAL NETWORK WITH MEMORY

WHAT ARE RNNS

The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps

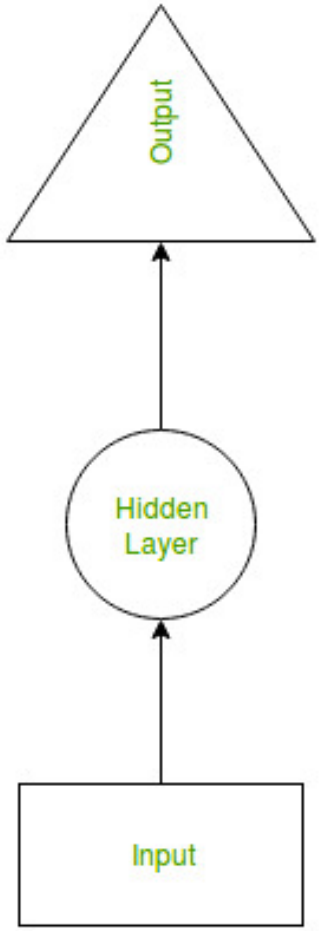
Examples of sequence data

	x		y
Speech recognition		→	"The quick brown fox jumped over the lazy dog."
Music generation	∅	→	
Sentiment classification	"There is nothing to like in this movie."	→	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger.

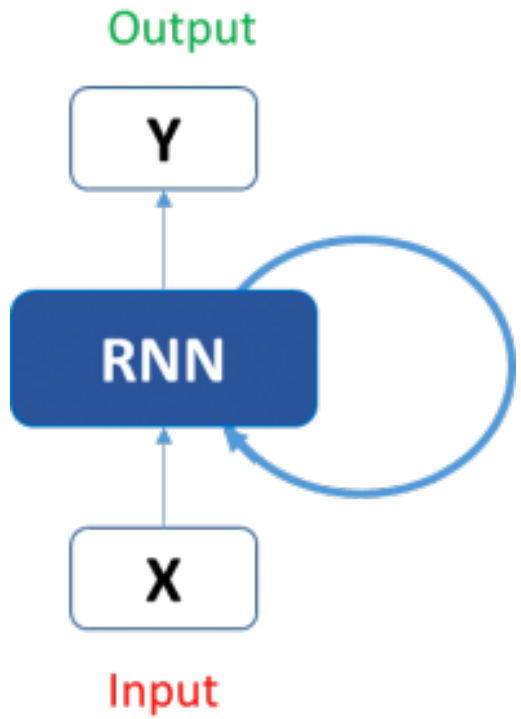
Andrew Ng

RECURRENT NEURON IN DETAIL

- **Recurrent Neural Network(RNN)** are a type of Neural Network where the **output from previous step are fed as input to the current step**. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is **Hidden state**, which remembers some information about a sequence.



RECURRENT NEURON IN DETAIL

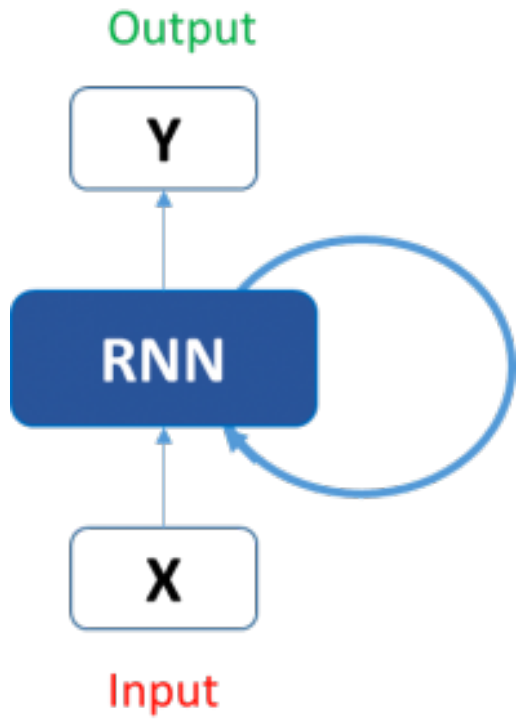


RECURRENT NEURON IN DETAIL

The formula for the current state can be written as –

$$h_t = f(h_{t-1}, x_t)$$

H_t is the new state
h_{t-1} is the previous state
x_t is the current input

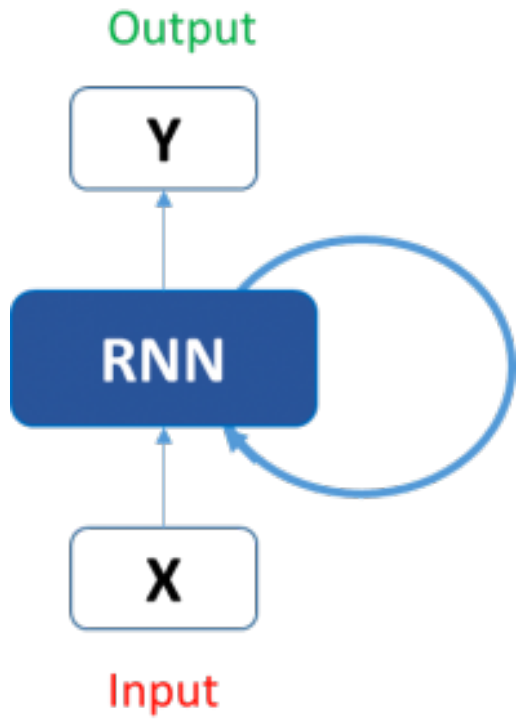


RECURRENT NEURON IN DETAIL

Taking the simplest form of a recurrent neural network, let's say that the
activation function is tanh
the weight at the recurrent neuron is W_{hh}
weight at the input neuron is W_{xh}

we can write the equation for the state at time t as –

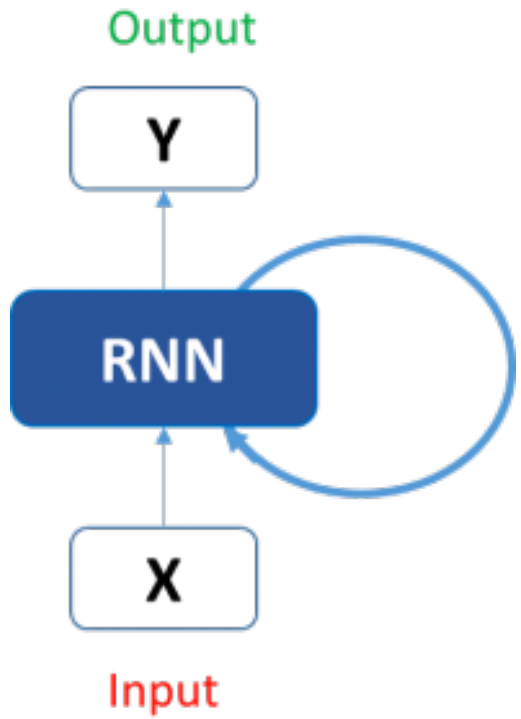
$$h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)$$



RECURRENT NEURON IN DETAIL

Now, once the current state is calculated we can calculate the output state as-

$$y_t = W_{hy} h_t$$



WHY DO WE NEED RNNS?

The limitations of the Neural network (CNNs)

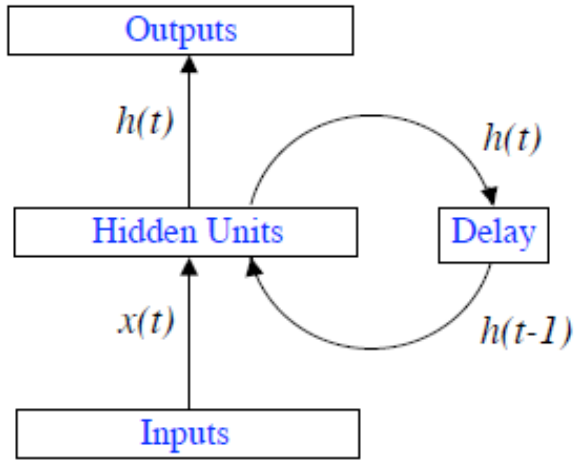
- Rely on the assumption of independence among the (training and test) examples.
  - After each data point is processed, the entire state of the network is lost
- Rely on examples being vectors of fixed length

We need to model the data with temporal or sequential structures and varying length of inputs and outputs

- Frames from video
- Snippets of audio
- Words pulled from sentences

WHAT ARE RNNS?

Recurrent neural networks (RNNs) are connectionist models with the ability to selectively pass information across sequence steps, while processing sequential data one element at a time.



The simplest form of ***fully recurrent neural network*** is an MLP with the previous set of hidden unit activations feeding back into the network along with the inputs

Allow a 'memory' of previous inputs to persist in the network's internal state, and thereby influence the network output

$$h(t) = f_H(W_{IH}x(t) + W_{HH}h(t-1))$$

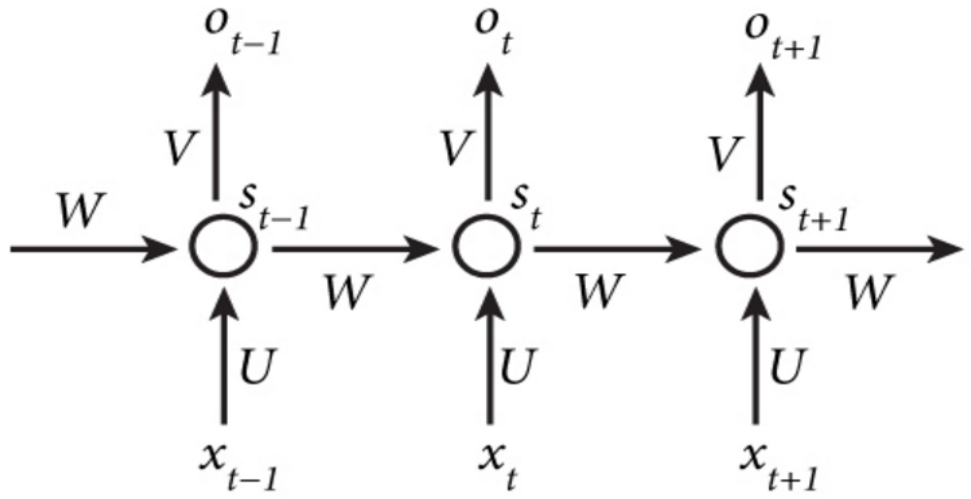
$$y(t) = f_O(W_{HO}h(t))$$

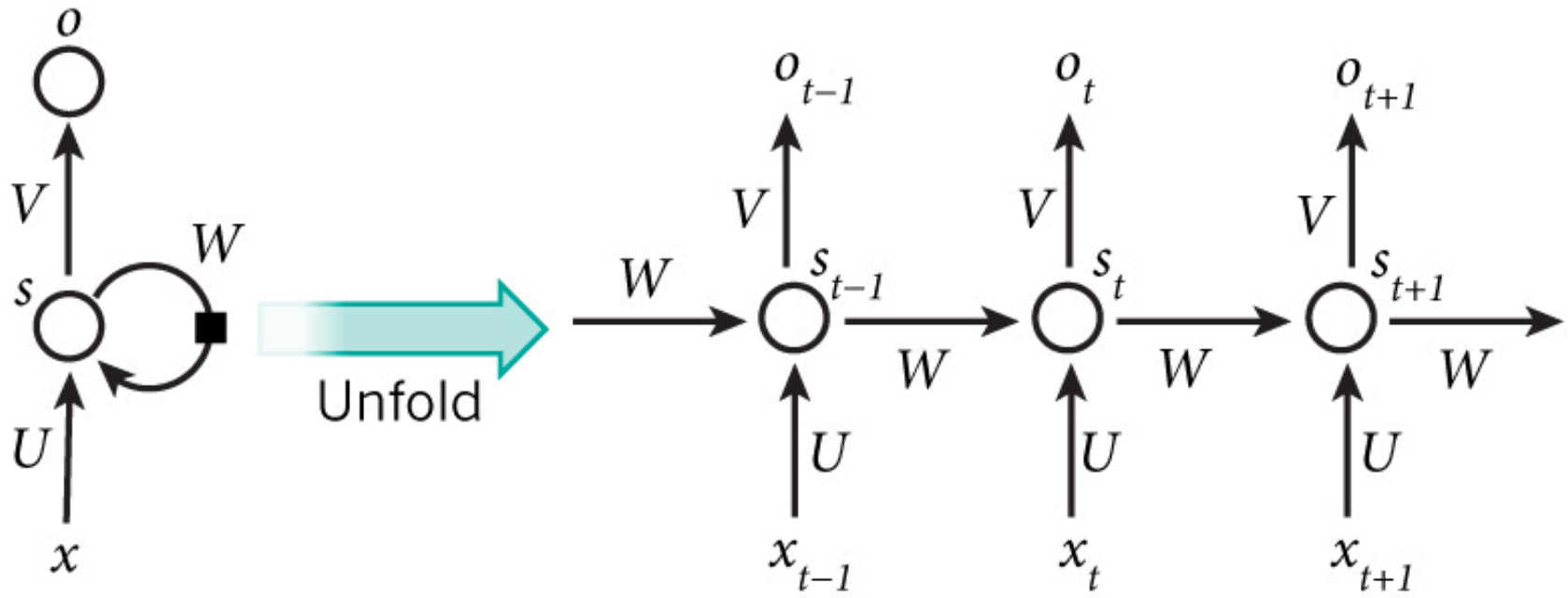
f_H and f_O are the activation function for hidden and output unit; W_{IH}, W_{HH}, and W_{HO} are connection weight matrices which are learnt by training

WHAT ARE RNNS?

- The recurrent network can be converted into a feed-forward network by ***unfolding over time***

An unfolded recurrent network. Each node represents a layer of network units at a single time step. The weighted connections from the input layer to hidden layer are labelled ‘w1’, those from the hidden layer to itself (i.e. the recurrent weights) are labelled ‘w2’ and the hidden to output weights are labelled‘w3’. Note that the same weights are reused at every time step. Bias weights are omitted for clarity.

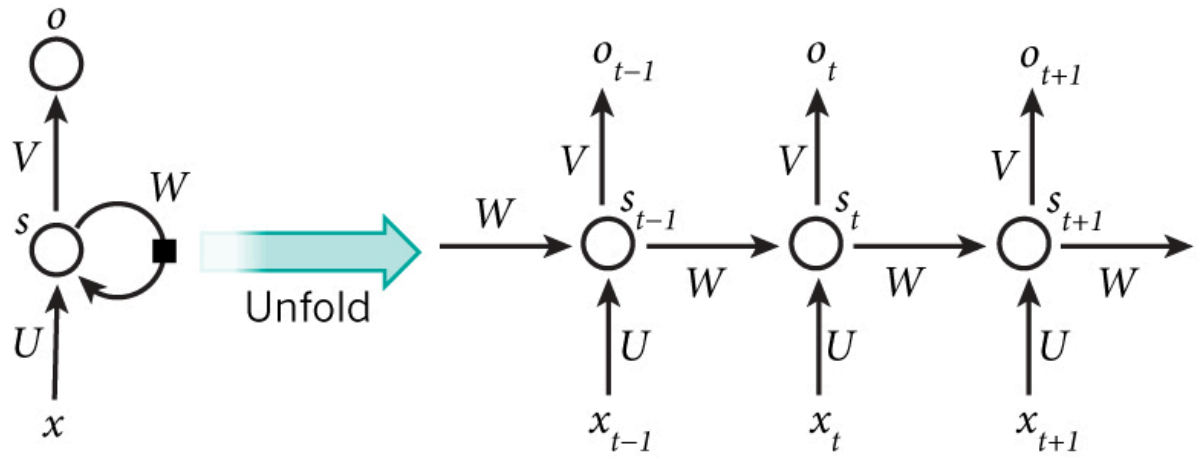




The above diagram shows a RNN being *unrolled* (or unfolded) into a full network. By unrolling we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word.

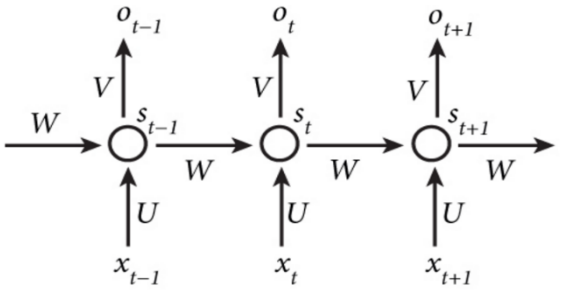
The formulas that govern the computation happening in a RNN are as follows:

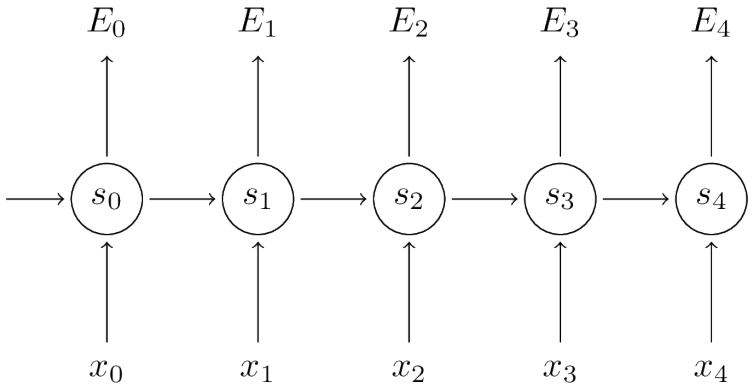
- x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the second word of a sentence.
- s_t is the hidden state at time step t. It's the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step: $s_t = f(Ux_t + Ws_{t-1})$. The function f usually is a nonlinearity such as tanh or ReLU. s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
- o_t is the output at step t. For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary. $o_t = \text{softmax}(Vs_t)$.



TRAINING RNNS (DETERMINE THE PARAMETERS)

Back Propagation Through Time (BPTT) is often used to learn the RNN
BPTT is an extension of the back-propagation (BP)





- The output of this RNN is $\hat{y}_t$

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = softmax(Vs_t)$$

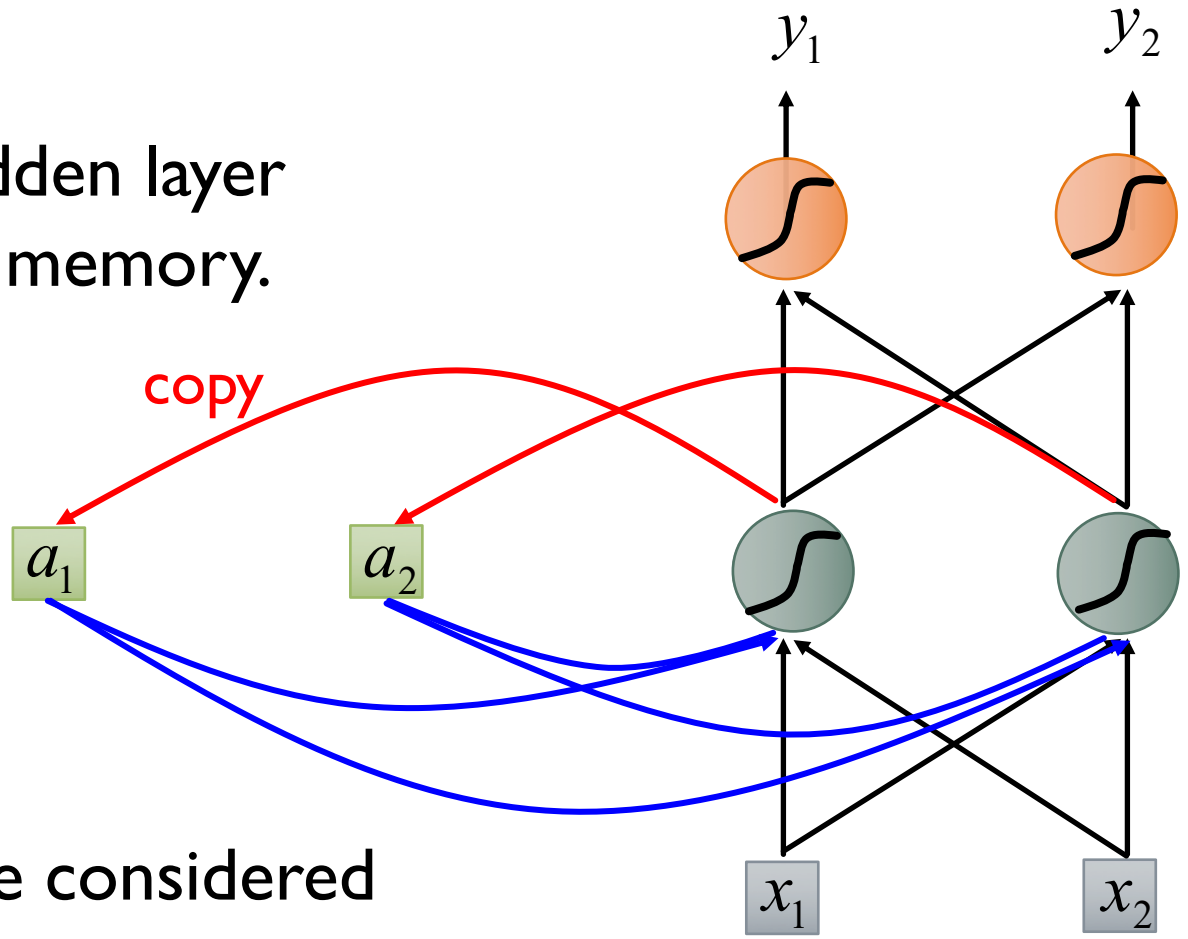
- The loss/error function of this network is

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$ → The error at each time step

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$ → the total loss is the sum of the errors at each time step

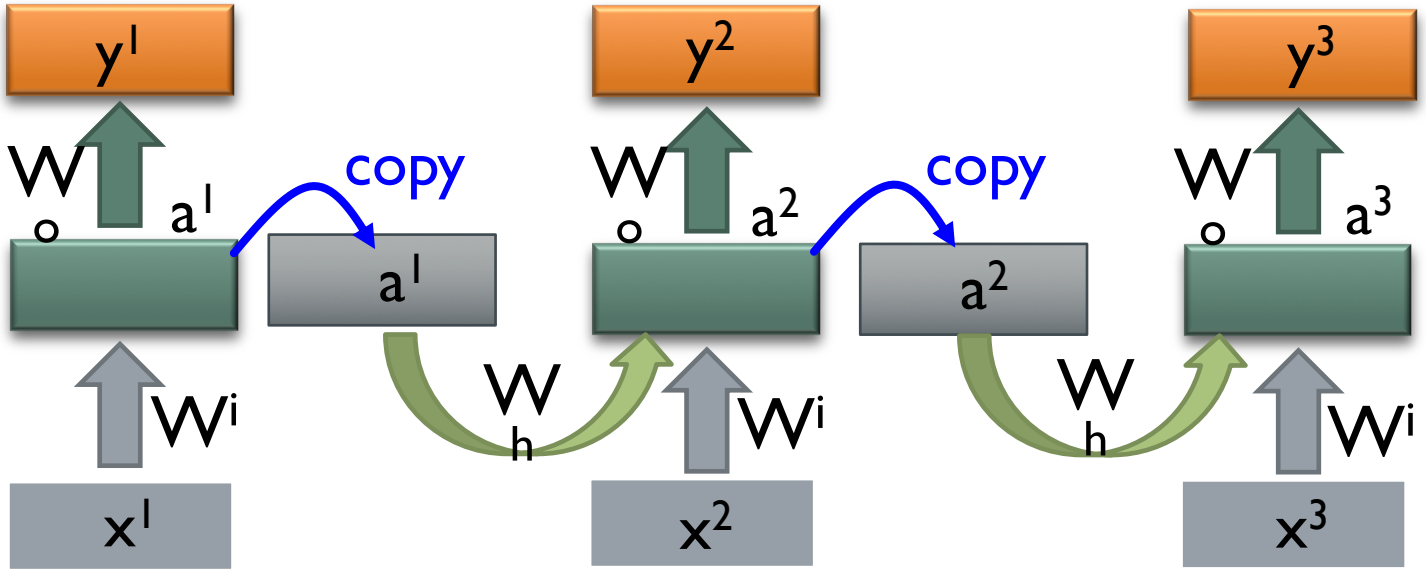
RECURRENT NEURAL NETWORK (RNN)

The output of hidden layer are stored in the memory.



Memory can be considered as another input.

RNN



The same network is used again and again.

Output y^i depends on $x^1, x^2, \ldots\ldots x^i$

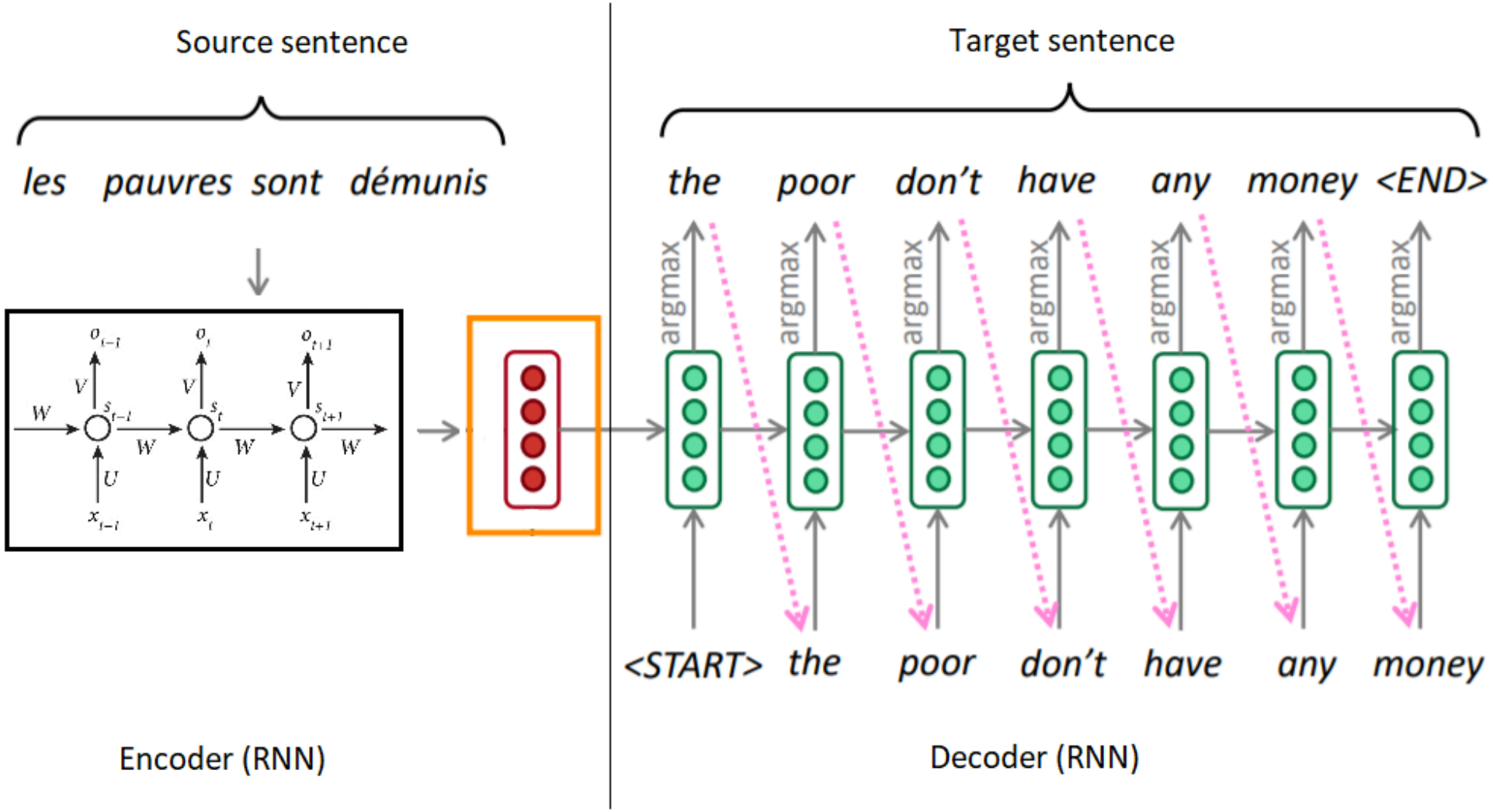
WHAT CAN RNNS DO?

- **Language Modelling and Generating Text**
- **Machine Translation**
- **Speech Recognition**
- **Generating Image Descriptions**

LANGUAGE MODELLING AND GENERATING TEXT

Text Generation is a type of Language Modelling problem. Language Modelling is the core problem for a number of of natural language processing tasks such as speech to text, conversational system, and text summarization. A trained language model learns the likelihood of occurrence of a word based on the previous sequence of words used in the text. Language models can be operated at character level, n-gram level, sentence level or even paragraph level.

LANGUAGE MODELLING AND GENERATING TEXT



MACHINE TRANSLATION

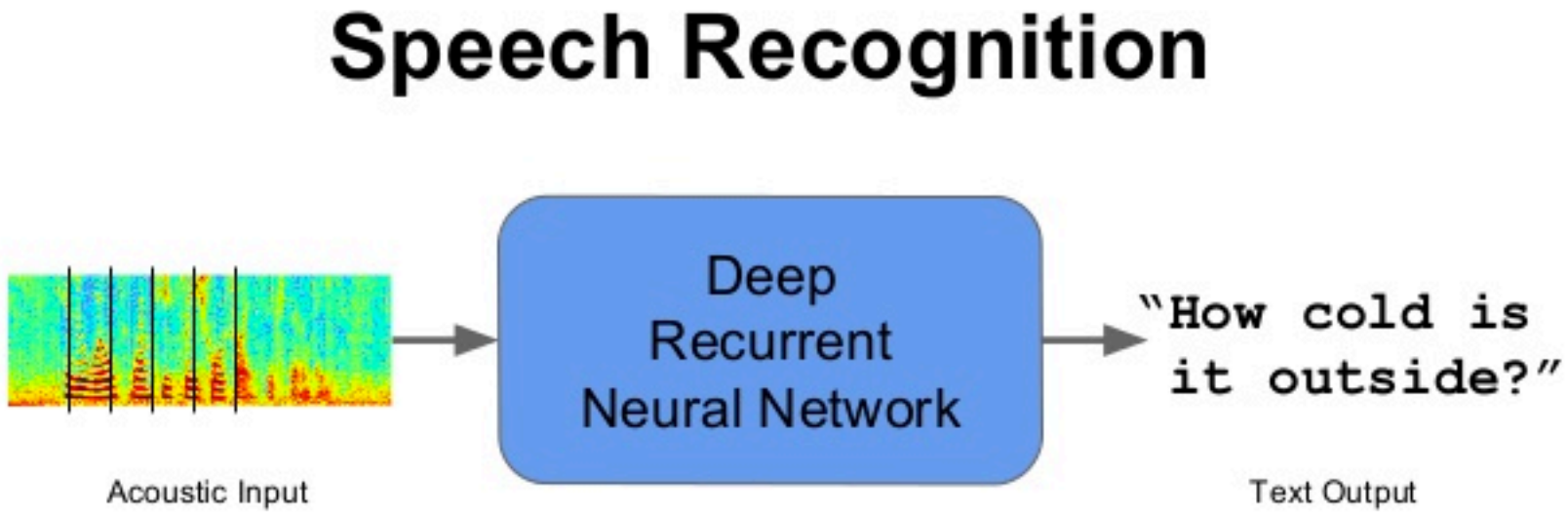
- The task of machine translation consists of reading text in one language and generating text in another language. When neural networks are used for this task, we talk about neural machine translation (NMT). Within NMT, the encoder-decoder structure is quite a popular RNN architecture

MACHINE TRANSLATION

- The task of machine translation consists of reading text in one language and generating text in another language. When neural networks are used for this task, we talk about neural machine translation (NMT). Within NMT, the encoder-decoder structure is quite a popular RNN architecture

SPEECH RECOGNITION

- Given an input sequence of acoustic signals from a sound wave, we can predict a sequence of phonetic segments together with their probabilities.

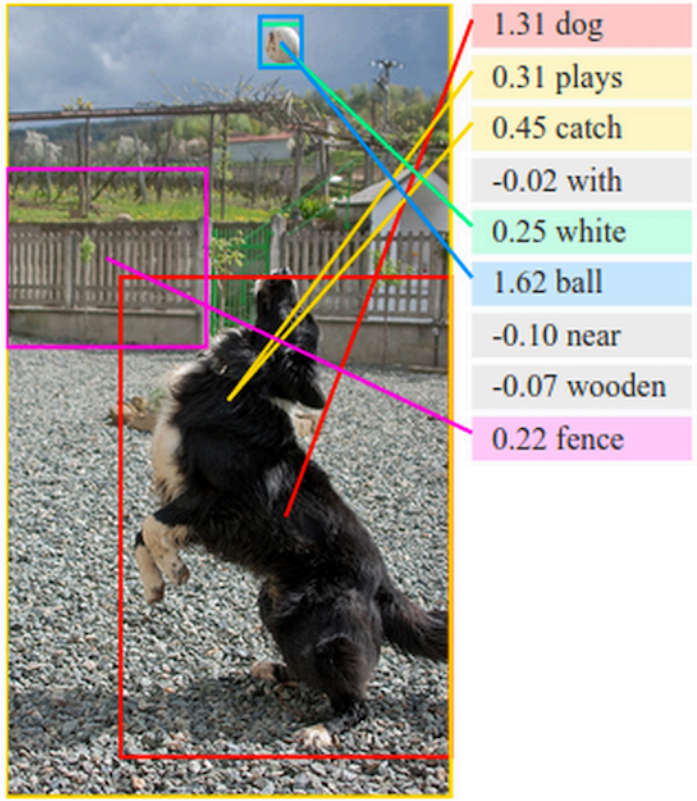


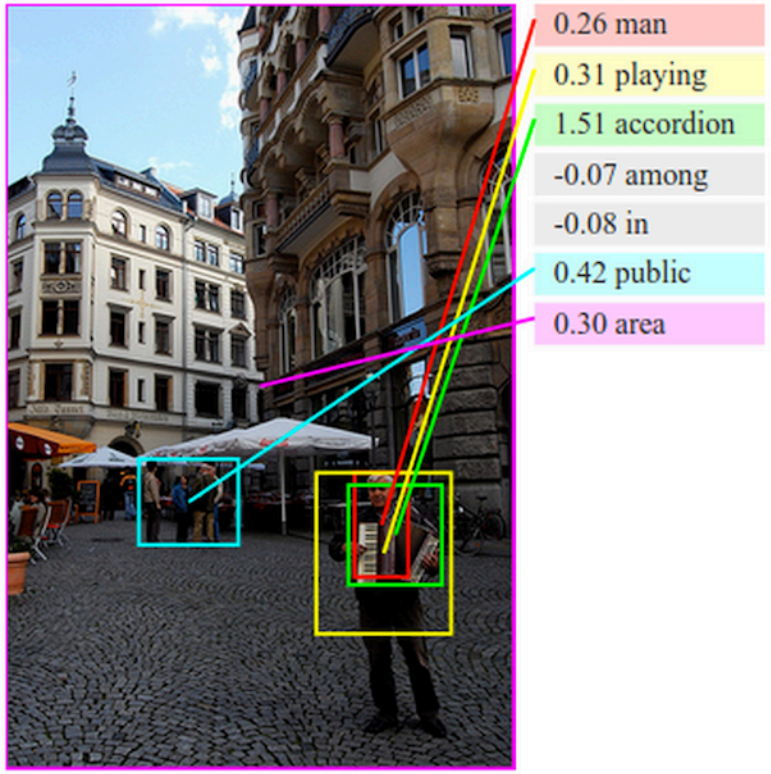
Reduced word errors by more than 30%

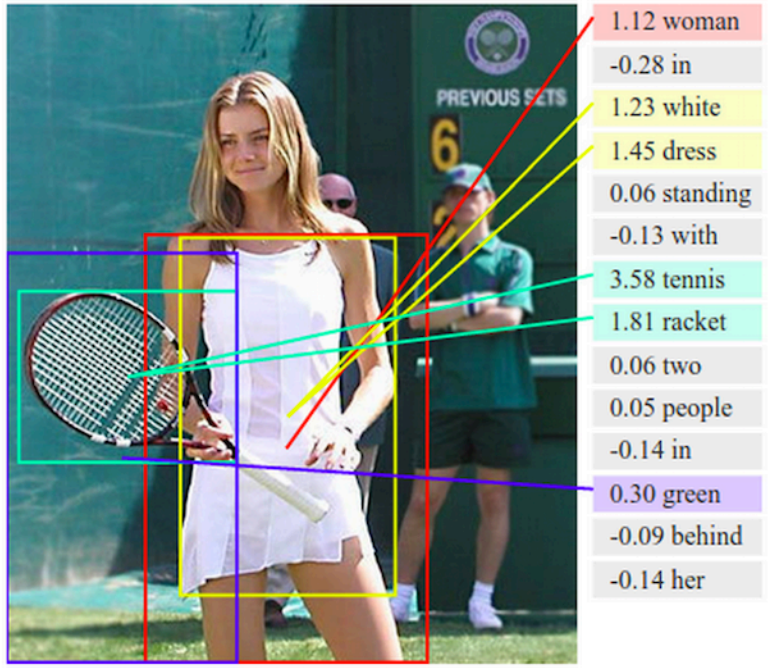
Google Research Blog - August 2012, August 2015

Research at Google

GENERATING IMAGE DESCRIPTIONS







WHAT ARE RNNS?

✓ The gradients of the error with respect to our parameters

Just like we sum up the errors, we also sum up the gradients at each time step for one training example. For parameter W, the gradient is

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

- Training RNNs (determine the parameters)

✓ The gradient at each time step

we use time 3 as an example

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W} \longrightarrow \text{Chain Rule}$$

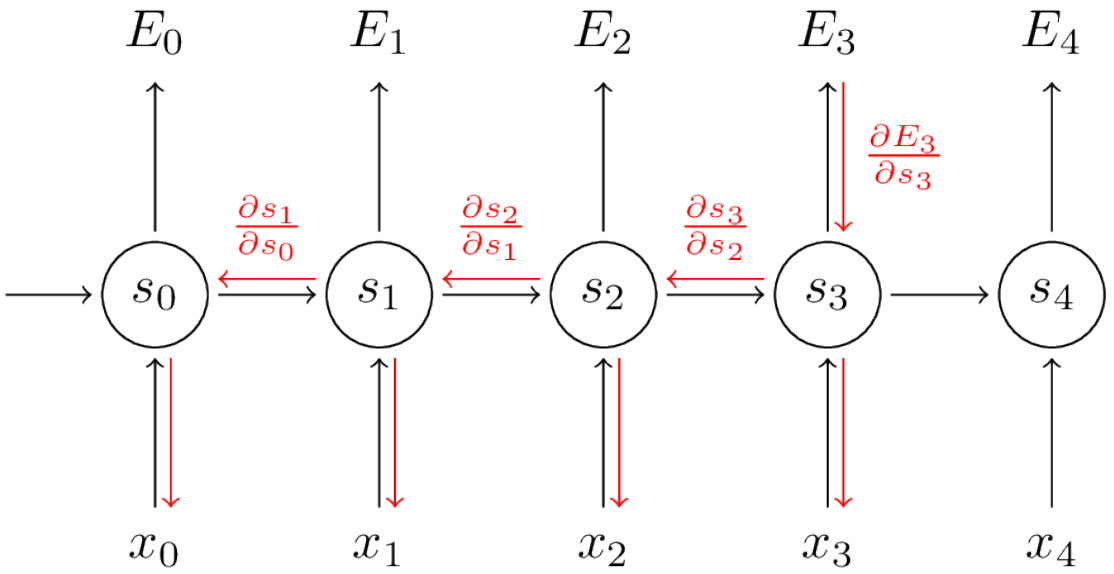
$$s_3 = \tanh(Ux_1 + Ws_2) \longrightarrow s_3 \text{ depends on } W \text{ and } s_1\text{, we cannot simply treat } s_2 \text{ a constant}$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W} \longrightarrow \text{Apply Chain Rule again on } s_k$$

TRAINING RNNS (DETERMINE THE PARAMETERS)

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

Becaise W is used in every step up to the output we care about, we need to back-propagate gradients from $t = 3$ through the network all the way to $t = 0$



THE VANISHING GRADIENT PROBLEM

To understand why, let's take a closer look at the gradient we calculated above:

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W} \longrightarrow \frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \frac{\partial s_k}{\partial W}$$

Because the layers and time steps of deep neural networks relate to each other through multiplication, derivatives are susceptible to vanishing

Gradient contributions from "far away" steps become zero, and the state at those steps doesn't contribute to what you are learning: You end up not learning long-range dependencies.

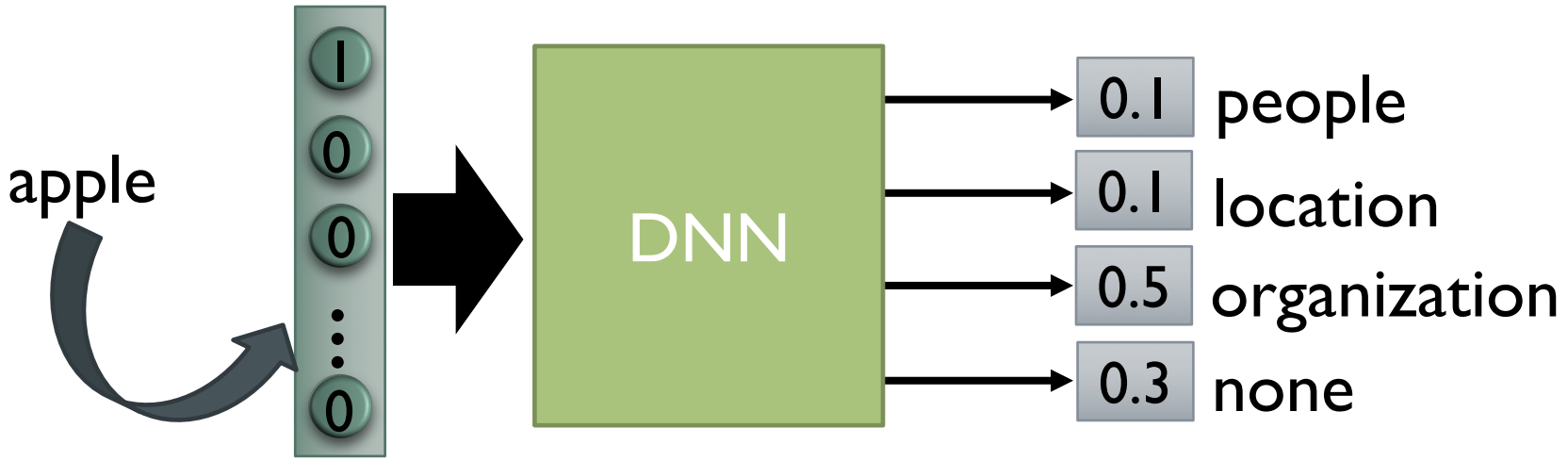
HOW TO SOLE THE VANISHING GRADIENT PROBLEM?

- Proper initialization of the W matrix can reduce the effect of vanishing gradients
- Use ReLU instead of tanh or sigmoid activation function *ReLU derivate is a constant of either 0 or 1, so it isn't likely to suffer from vanishing gradients*
- Use Long Short-Term Memory or Gated Recurrent unit architectures

NEURAL NETWORK NEEDS MEMORY

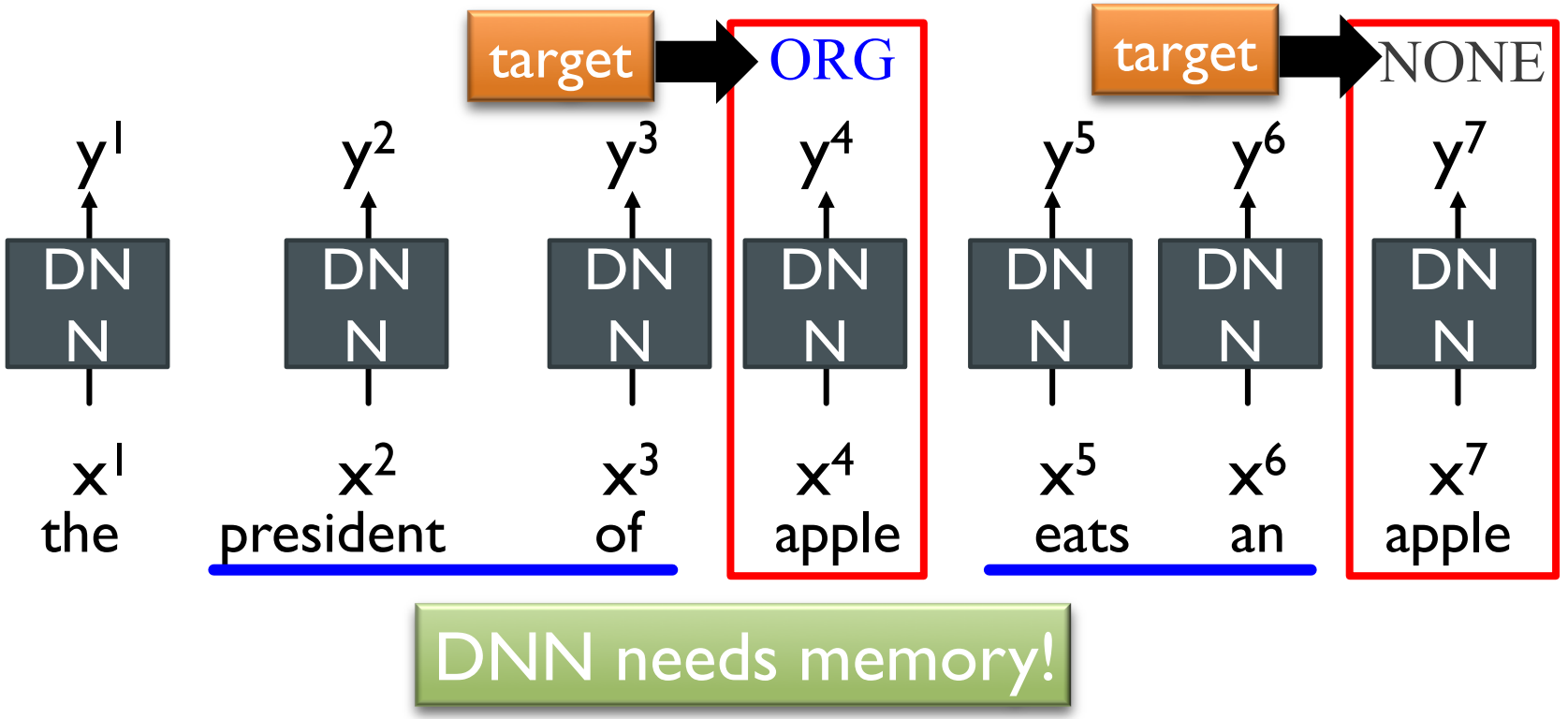
Name Entity Recognition

- Detecting named entities like name of people, locations, organization, etc. in a sentence.

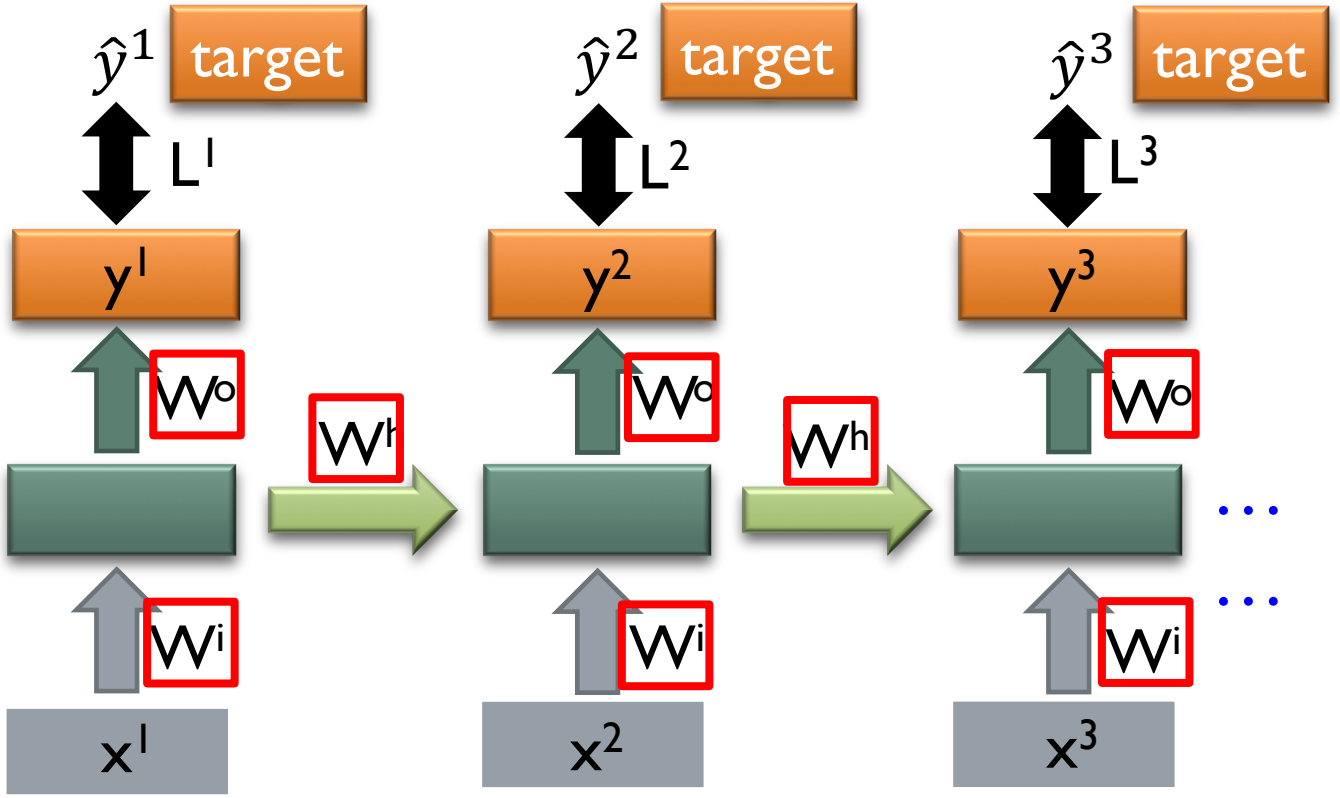


NEURAL NETWORK NEEDS MEMORY

- Name Entity Recognition
  - Detecting named entities like name of people, locations, organization, etc. in a sentence.



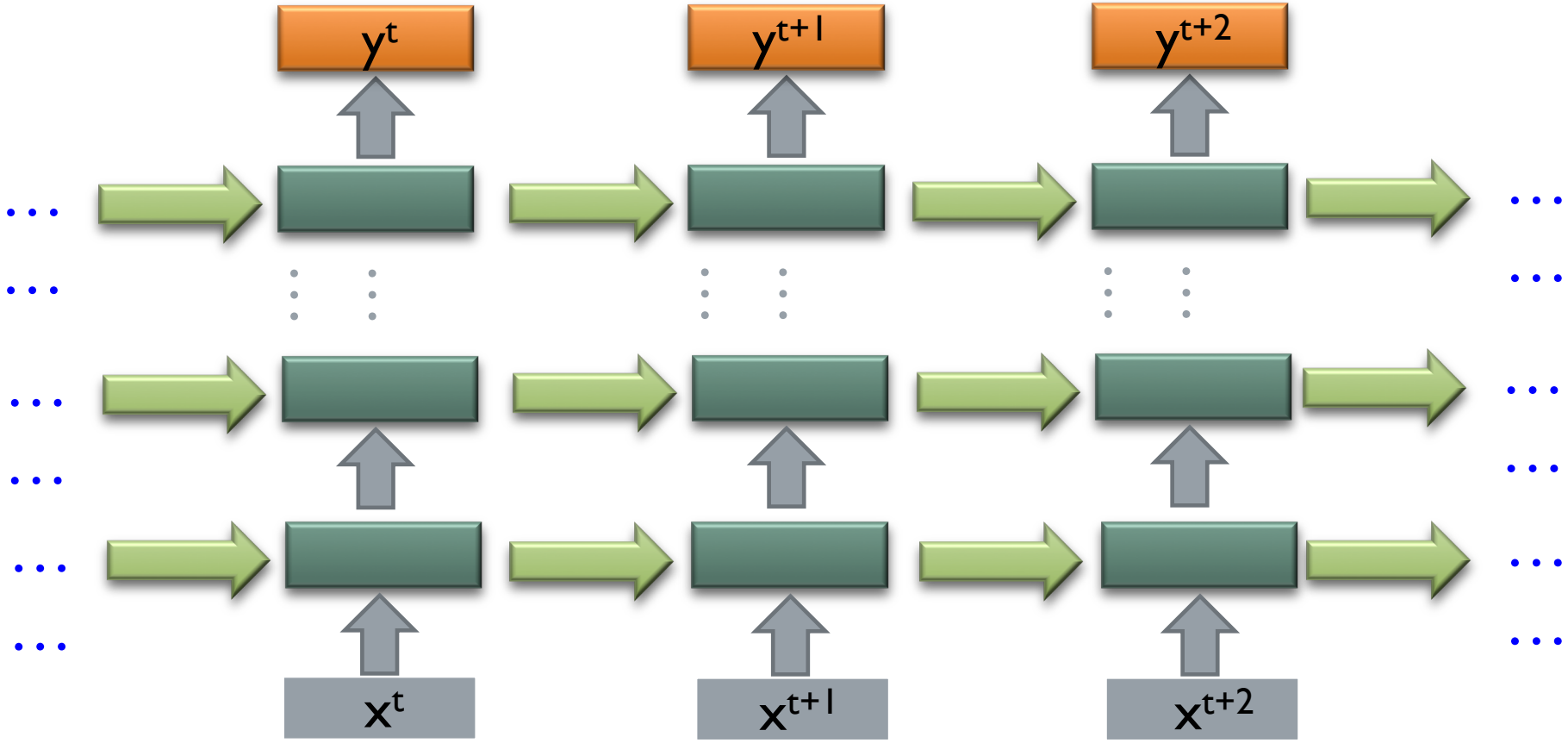
How to train?



Find the network parameters to minimize the total cost:

Backpropagation through time (BPTT)

OF COURSE IT CAN BE DEEP …



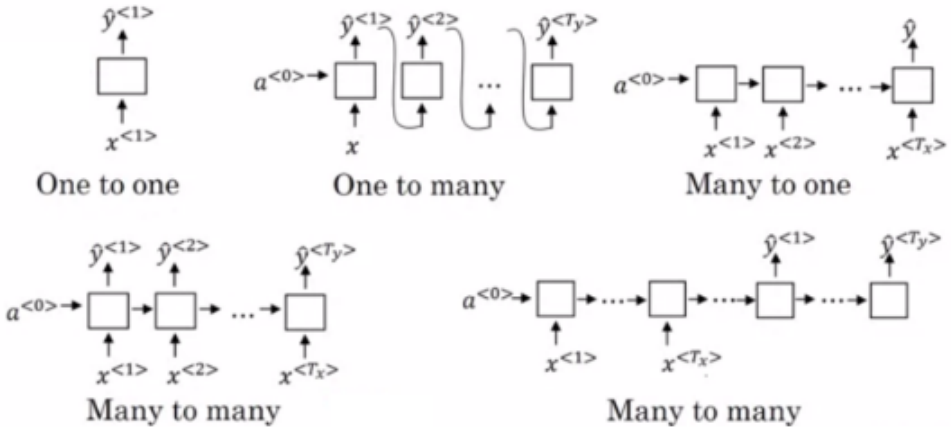
TYPES OF RNN

- **one to one** - Here we have single input and single output. This is popularly known as a Vanilla Neural Network.
- **one to many** - Here we have single input and multiple outputs. A good example where one to many RNN is used is for Image Captioning.
- **many to one** - Here we have multiple inputs and single output. An example of this would be sentiment prediction.
- **many to many** - Here we have multiple inputs as well as outputs. Machine translation is on such example where you give in a sequence of inputs in one language and it generates your output in different languages.

TYPES OF RNN

- **One to One RNN**

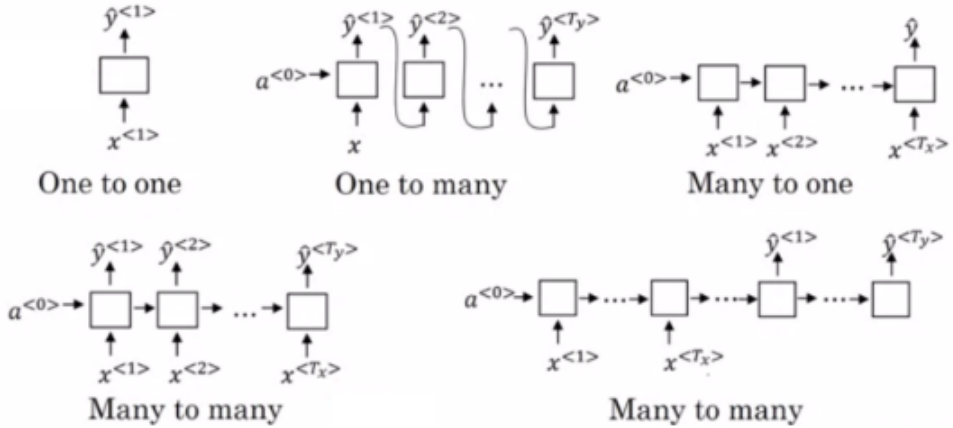
One to One RNN(Tx= Ty=1) is the most basic and traditional form of Neural Network, as you can see in the above picture, giving a single output for a single input.



TYPES OF RNN

- **One to Many**

One to Many (Tx=1, Ty>1) is a kind of RNN architecture that is implemented in situations where multiple output is given for a single input. Music generation will be a reference example of its application. RNN models are used in Music generation models to produce a piece of music (multiple output) from a single musical note (single input).

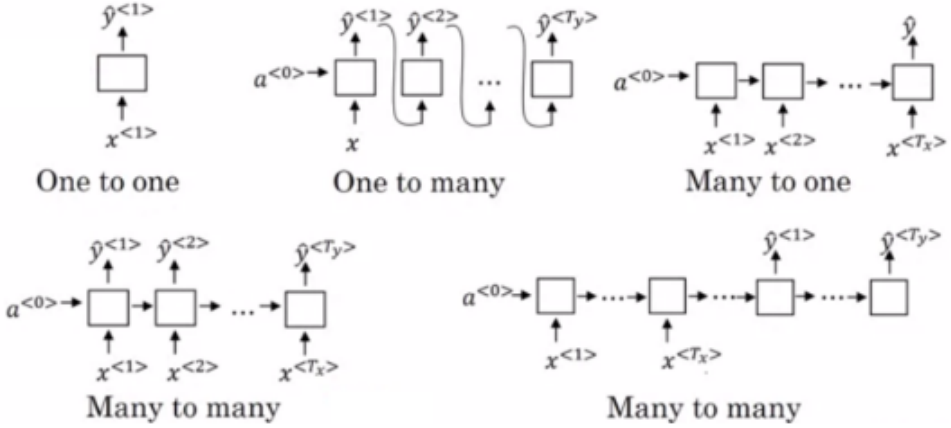


TYPES OF RNN

- **Many to One**

Many–to–one architecture of the RNN (Tx>1, Ty=1) is generally used as a common example for model of sentiment analysis. This type of model is used, as the name implies, when multiple inputs are needed to provide one output.

For example: The model for analyzing Twitter sentiment. A text entry (words as multiple inputs) in that model gives its set feeling (single output). One example this may be the model of film ratings this uses review texts as input to rate a film that can range from 1 to 5.



TYPES OF RNN

- **Many — to — Many**

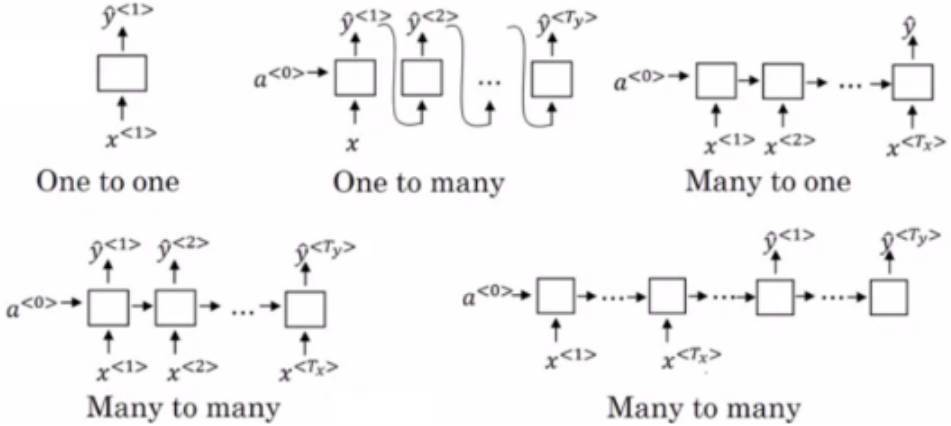
Many — to — many RNN architectures (Tx>1, Ty>1) take multiple inputs and offer multiple outputs, but many — to — many models can be two types as shown above:

- **Tx = Ty:**

That is the position where it applies to the case where layers of input and output are of the same thickness. It can also be interpreted as any output information can be found in Named–Entity Recognition.

- **Tx ! = Ty :**

Many–to–architecture can also be represented in models where input and layers are of different sizes, and Machine Translation demonstrates the most common application of this type of RNN architecture. As a consequence, machine translation models are able to return words more or less than the input string due to a non – equivalent number–to–many RNN architecture that operates in the context.



- Implementing a simple RNN
  Adding Long Short-Term Memory (LSTM)
  Using gated recurrent units (GRUs)
  Implementing bidirectional RNNs
  Character-level text generation
  Tensorflow Recurrent Neural Network (LSTM)-MNIST
  Tensorflow Bi-directional Recurrent Neural Network (LSTM)-MNIST
- Tensorflow Dynamic Recurrent Neural Network (LSTM)

PROBLEMS WITH RNN : **SHORT-TERM MEMORY**

- Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning.
- During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weights. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn't contribute too much learning.

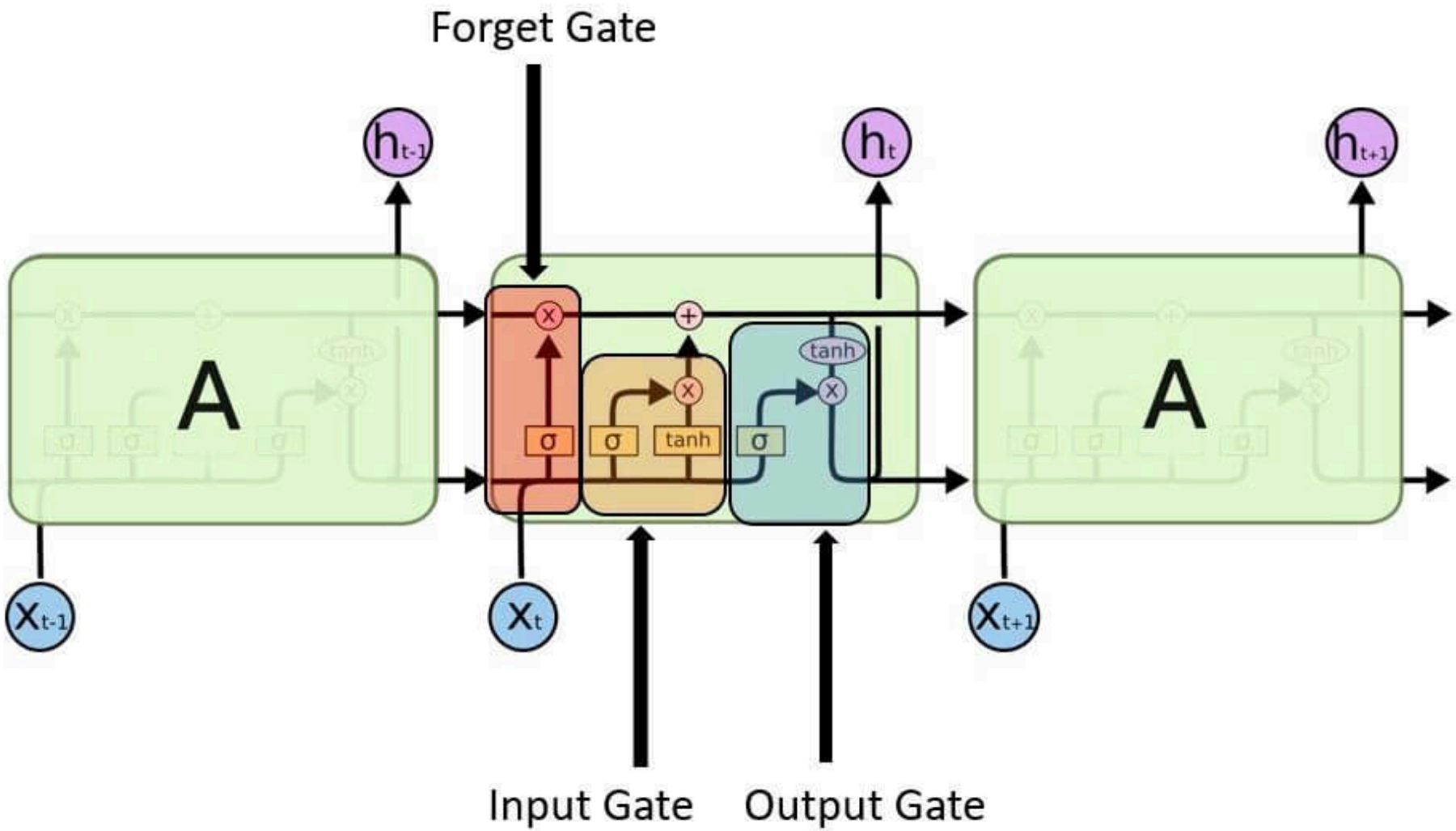
LSTM'S AND GRU'S AS A SOLUTION

- LSTM 's and GRU's were created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information.

ADDING LONG SHORT-TERM MEMORY (LSTM)

- Long Short Term Memory is a kind of recurrent neural network. In RNN output from the last step is fed as input in the current step. LSTM was desgined by Hochreiter & Schmidhuber. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give efficient performance. LSTM can by default retain the information for long period of time. It is used for processing, predicting and classifying on the basis of time series data.

- A general **LSTM** unit is composed of a cell, an input gate, an output gate, and a forget gate. The cell remembers values over arbitrary time intervals, and three gates regulate the flow of information into and out of the cell. LSTM is well-suited to classify, process, and predict the time series given of unknown duration.


Forget Gate
h_{t-1}
h_t
h_{t+1}
A
A
σ
σ
tanh
σ
tanh
X_{t-1}
X_t
X_{t+1}
Input Gate
Output Gate

Input gate- It discover which value from input should be used to modify the memory. **Sigmoid** function decides which values to let through 0 or 1. And **tanh** function gives weightage to the values which are passed, deciding their level of importance ranging from **-1** to **1**.

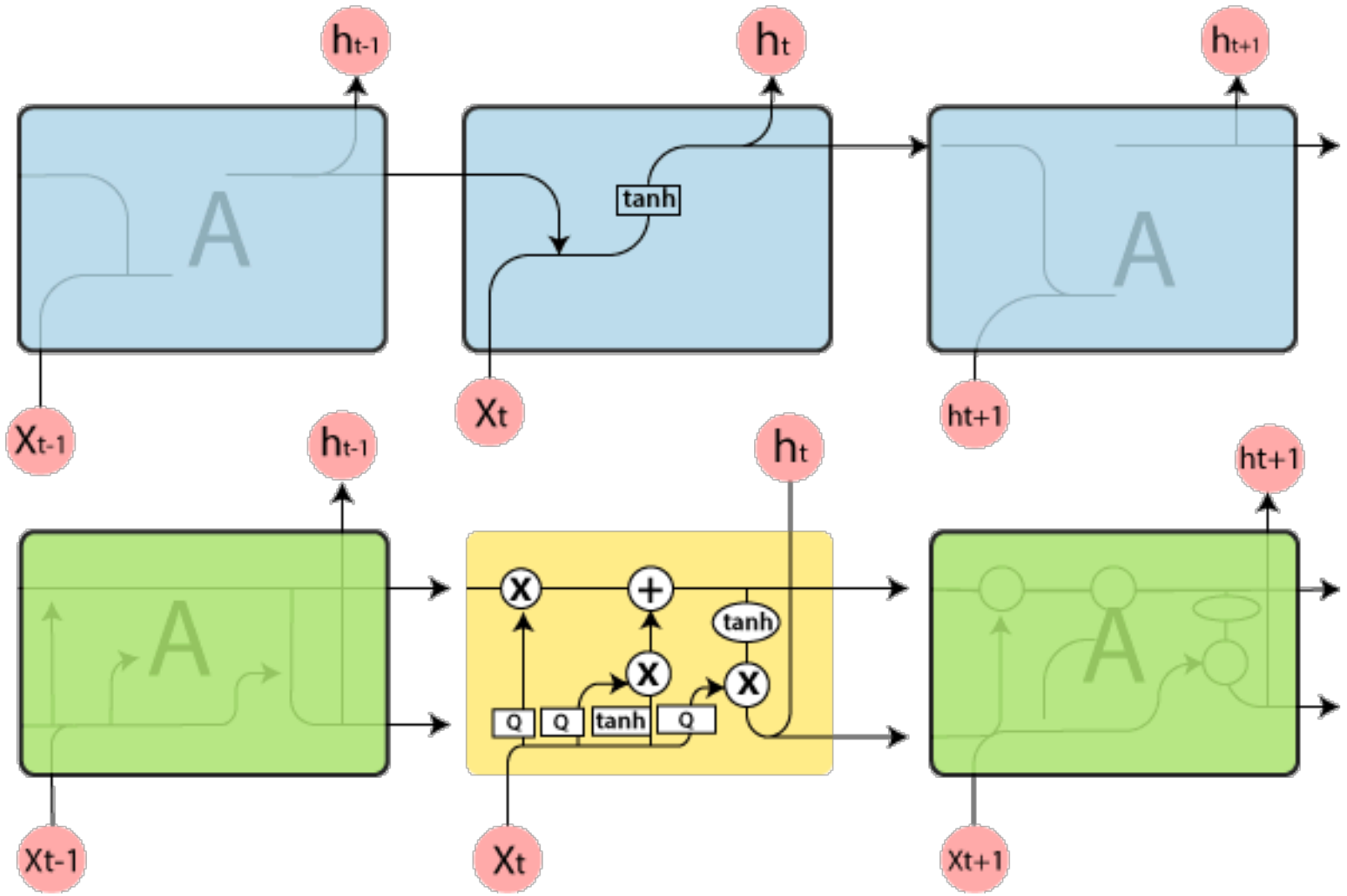
$$i_t = \sigma(W_i.[h_t - 1, x_t] + bi)$$
$$C_t = \tanh(W_C.[h_t - 1, x_t] + b_C)$$

Forget gate- It discover the details to be discarded from the block. A sigmoid function decides it. It looks at the previous state **(ht-1)** and the content input (Xt) and outputs a number between 0(omit this) and 1(keep this) for each number in the cell state **Ct-1**

$$f_t = \sigma(W_f.[h_t - 1, x_t] + b_f)$$

Output gate- The input and the memory of the block are used to decide the output. Sigmoid function decides which values to let through 0 or 1. And **tanh** function decides which values to let through 0, 1. And tanh function gives weightage to the values which are passed, deciding their level of importance ranging from **-1 to 1** and multiplied with an output of **sigmoid**.

$$O_t = \sigma(W_o[h_t - 1, x_t] + b_o$$
$$h_t = o_t * \tanh(C_t)$$

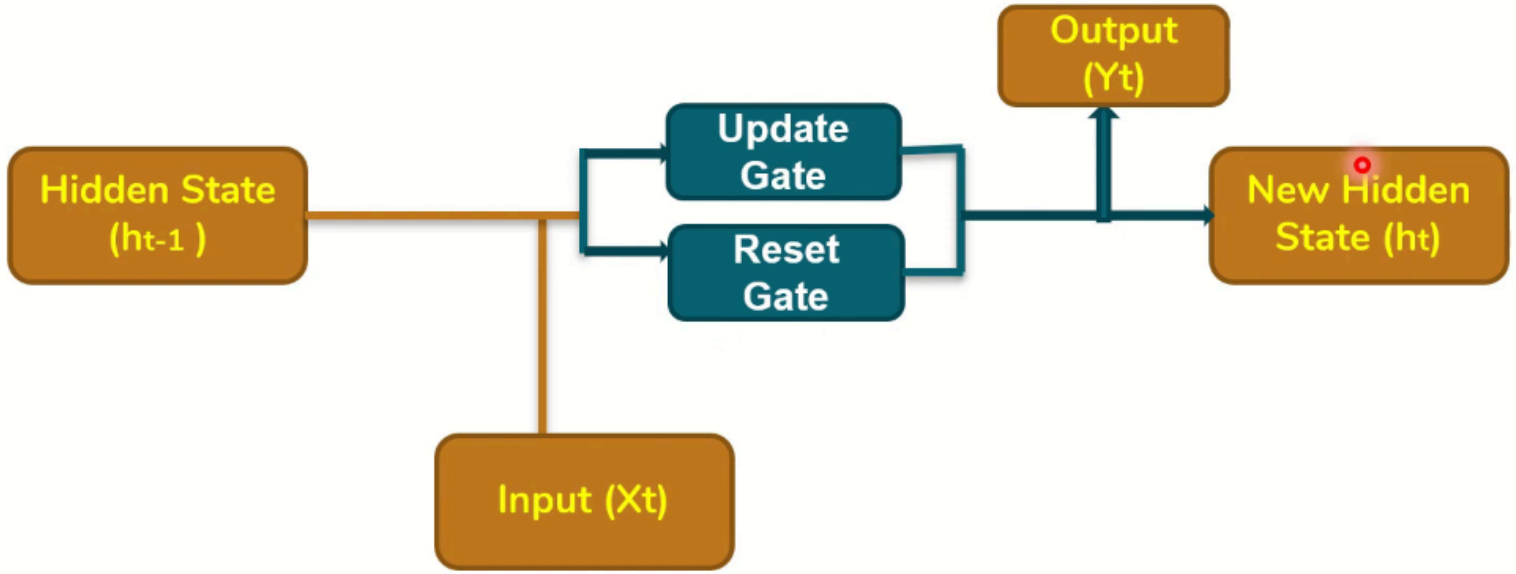

ht-1
ht
ht+1
A
tanh
A
Xt-1
Xt
ht+1
ht-1
ht
ht+1
A
X
+
tanh
X
X
Q
Q
tanh
Q
A
Xt-1
Xt
Xt+1

GATED RECURRENT UNITS (GRU)

- Introduced by Cho, et al. in 2014, GRU (Gated Recurrent Unit) aims to solve the **vanishing gradient problem** which comes with a standard recurrent neural network. GRU can also be considered as a variation on the LSTM because both are designed similarly and, in some cases, produce equally excellent results.

GATED RECURRENT UNITS (GRU)

- The idea is pretty much similar. The gates help in determining what to be retained/passed or dropped.
- The gates shall get the value between 0 and 1 as we have seen earlier. A 0 from the gate is revealing that the data is unimportant. 1 says it is important (It is like closer to 0, it is unimportant, closer to 1 it is important).
- GRUs do not have the Cell State and Hidden states are used!



GATED RECURRENT UNITS (GRU)

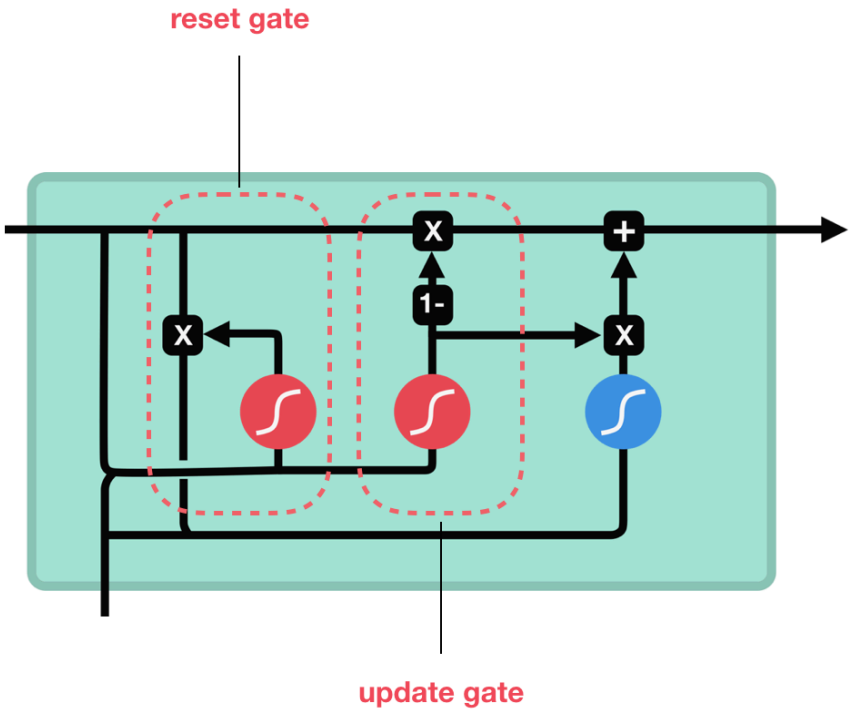
To solve the Vanishing-Exploding gradients problem often encountered during the operation of a basic Recurrent Neural Network, many variations were developed. One of the most famous variations is the **Long Short Term Memory Network(LSTM)**. One of the lesser known but equally effective variations is the **Gated Recurrent Unit Network(GRU)**.

Unlike LSTM, it consists of only three gates and does not maintain an Internal Cell State. The information which is stored in the Internal Cell State in an LSTM recurrent unit is incorporated into the hidden state of the Gated Recurrent Unit. This collective information is passed onto the next Gated Recurrent Unit. The different gates of a GRU are :-

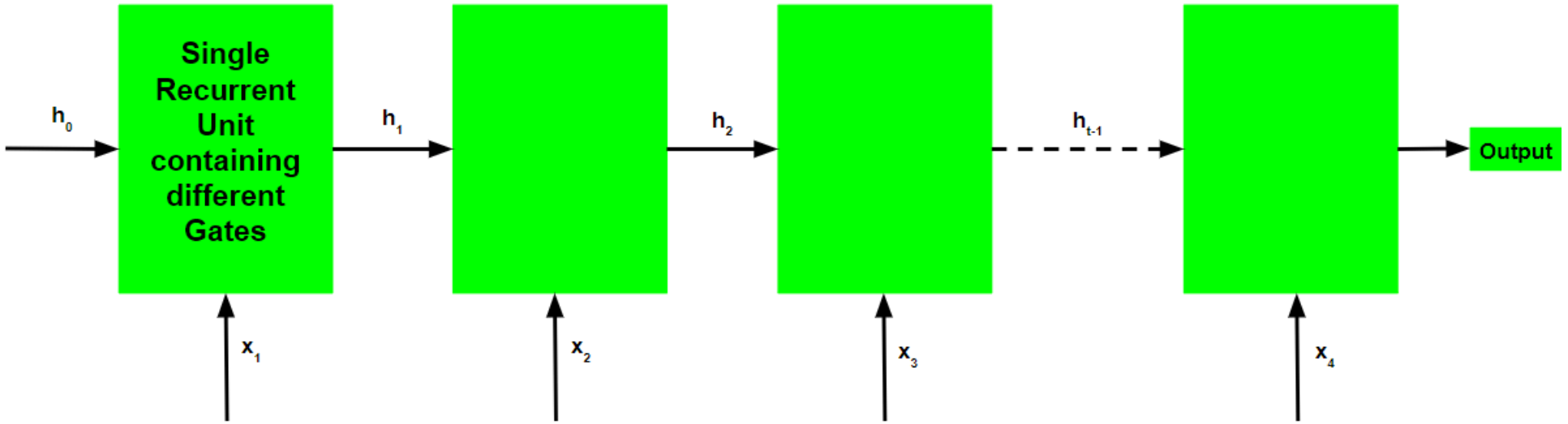
GATED RECURRENT UNITS (GRU)

1.Update Gate(z): It determines how much of the past knowledge needs to be passed along into the future. It is analogous to the Output Gate in an LSTM recurrent unit.
2.Reset Gate(r): It determines how much of the past knowledge to forget. It is analogous to the combination of the Input Gate and the Forget Gate in an LSTM recurrent unit.

3.Current Memory Gate: $\bar{h}_t$ It is often overlooked during a typical discussion on Gated Recurrent Unit Network. It is incorporated into the Reset Gate just like the Input Modulation Gate is a sub-part of the Input Gate and is used to introduce some non-linearity into the input and to also make the input Zero-mean. Another reason to make it a sub-part of the Reset gate is to reduce the effect that previous information has on the current information that is being passed into the future.



- The basic work-flow of a Gated Recurrent Unit Network is similar to that of a basic Recurrent Neural Network when illustrated, the main difference between the two is in the internal working within each recurrent unit as Gated Recurrent Unit networks consist of gates which modulate the current input and the previous hidden state.



WORKING OF A GATED RECURRENT UNIT:

1. Take input the current input and and the previous hidden state as vectors.
2. Calculate the values of the three different gates by following the steps given below:-
3. For each gate, calculate the parameterized current input and previous hidden state vectors by performing element-wise multiplication (hadmard product) between the concerned vector and the respective weights for each gate.
4. Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.

Update Gate : Sigmoid Function
Reset Gate : Sigmoid Function

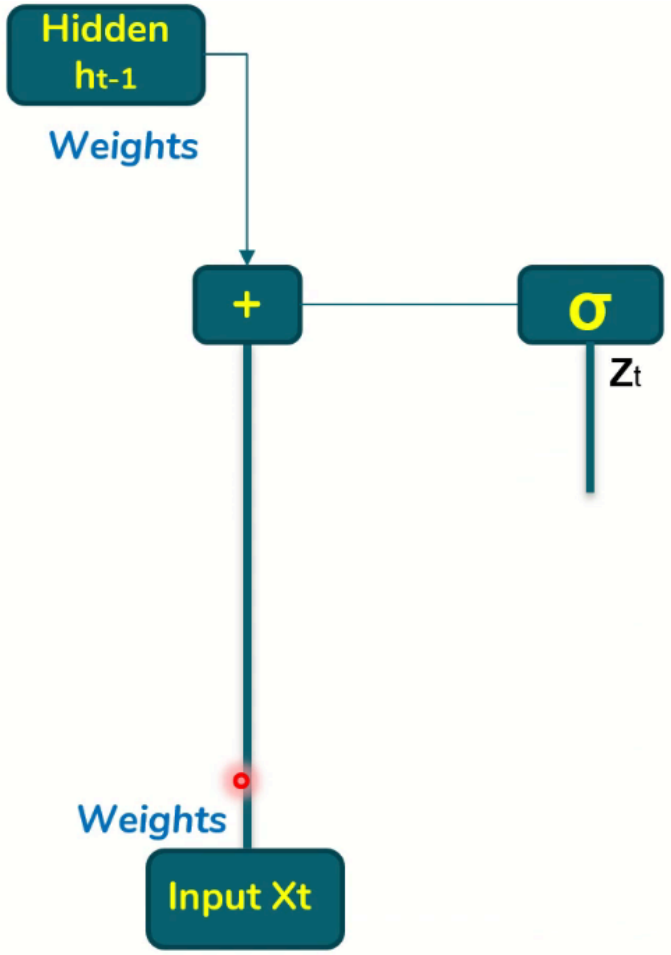
- The process of calculating the Current Memory Gate is a little different. First, the Hadmard product of the Reset Gate and the previous hidden state vector is calculated. Then this vector is parameterized and then added to the parameterized current input vector.

$$\overline{h}_t = tanh(W \odot x_t + W \odot (r_t \odot h_{t-1}))$$

- To calculate the current hidden state, first a vector of ones and the same dimensions as that of the input is defined. This vector will be called ones and mathematically be denoted by 1. First calculate the hadmard product of the update gate and the previous hidden state vector. Then generate a new vector by subtracting the update gate from ones and then calculate the hadmard product of the newly generated vector with the current memory gate. Finally add the two vectors to get the current hidden state vector.

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \overline{h}_t$$

STEP- I



$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

STEP-2

- **Reset gate:**

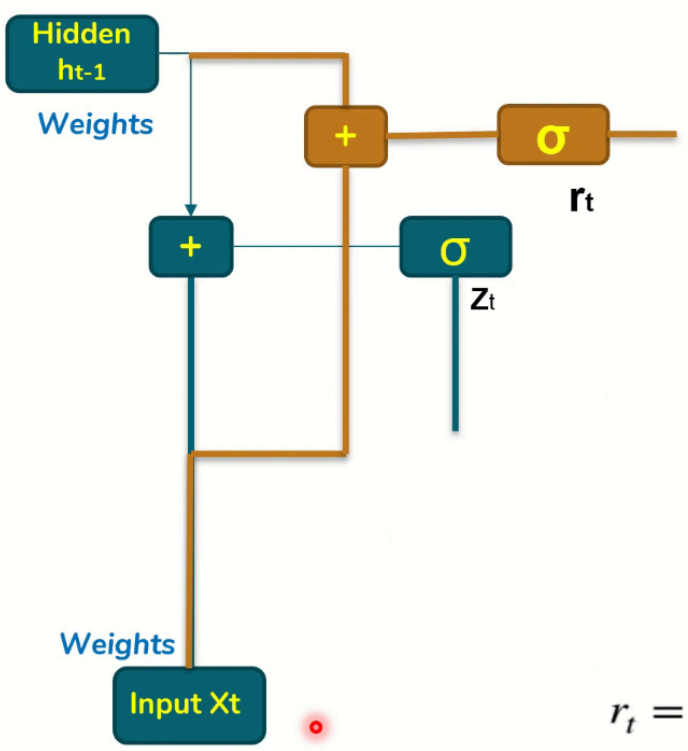
- Represented by r_t

- Input represented by X_t, Previous h_{t-1} State information multiplied with respective weights as the parameters. (Notations differ, nothing to worry as long as you understand that respective weights are to be used appropriately)

- Sigmoid activation is used to derive r_t.

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

STEP-2



$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

STEP-3

A New Component is getting introduced now!! ☺ Yes, it is a memory component and will use the reset gate to get the relevant information stored from the past.

You are reviewing a movie. Initially you start with " the movie is directed by X, it has featured Y, Music by Z, etc.. After about 10 lines, you say, the movie, I think is a bad one for the money I paid".

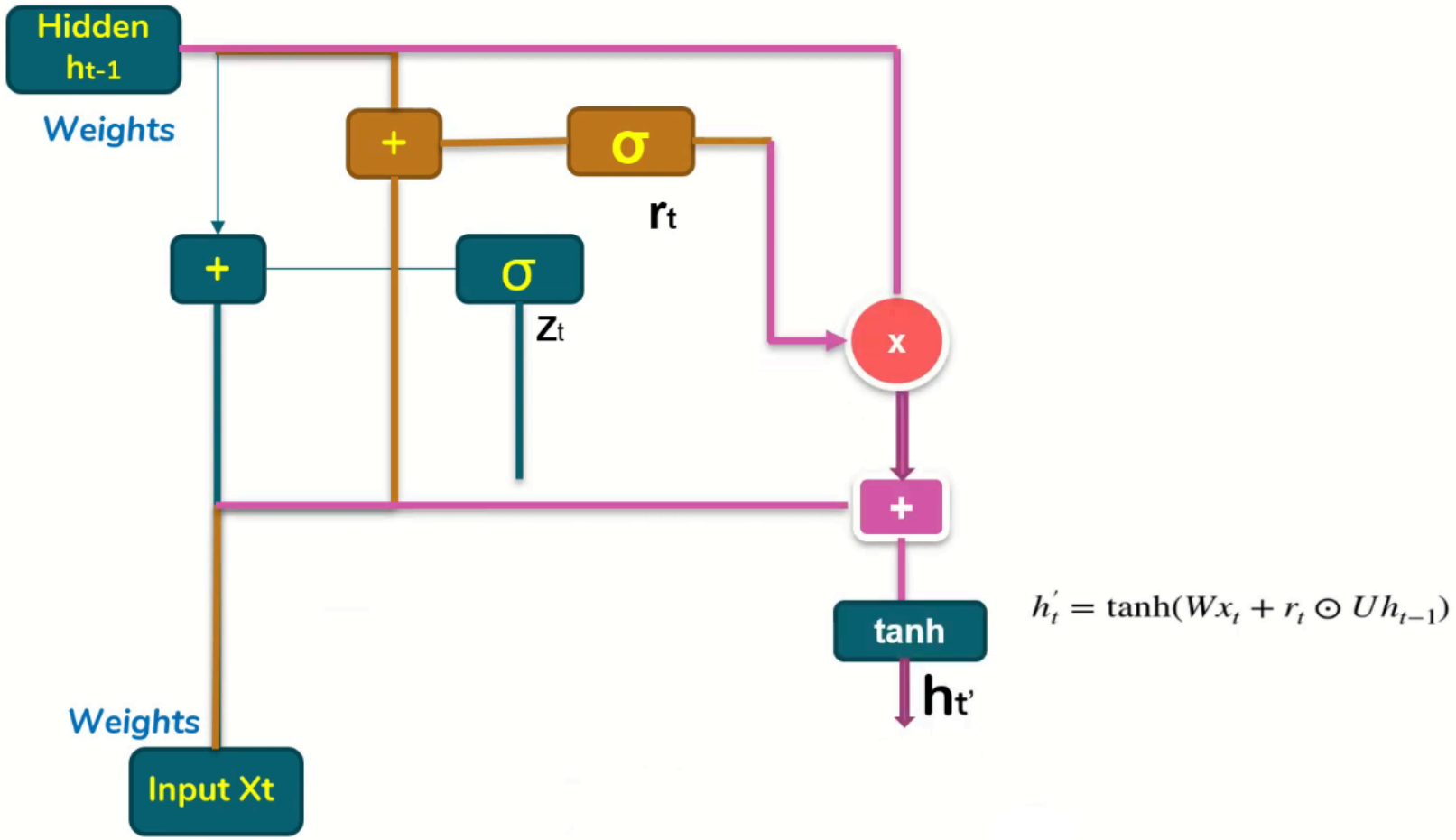
So, which is the actual review? The final line. So, the neural network should not remember the past sentences and focus on the last sentence to get the crux. It is what is enabled here.

Here, r_t will be assigned 0 until the final phrase is in analyze so as to drop the unfocussed items.

Then comes the tanh activation and you get h_t'

$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

STEP-3



$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

STEP-4

- As the last step, the network needs to calculate h_t — vector which holds information for the current unit and passes it down to the network.
- In order to do that the update gate is needed.

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t^{'}$$

Thanks